UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P O Box 1450
Alexandria, Virginia 22313-1450
www uspto gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/717,941	11/20/2003	Stephen La Roux Blinick	TUC920030135US1	9013

45216 7590 08/12/2009
Kunzler & McKenzie
8 EAST BROADWAY
SUITE 600
SALT LAKE CITY, UT 84111

EXAMINER	
WANG, BEN C	

ART UNIT	PAPER NUMBER
2192	

MAIL DATE	DELIVERY MODE
08/12/2009	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

	Application No.	Applicant(s)	
Office Action Summary	10/717,941	BLINICK ET AL.	
	Examiner	**Art Unit**	
	BEN C. WANG	2192	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *18 May 2009*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-3,9-16,18,20-23 and 28-36* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-3, 9-16, 18, 20-23, and 28-36* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)

2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)

3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.

4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .

5) ☐ Notice of Informal Patent Application

6) ☐ Other: _____.

DETAILED ACTION

1. A request for continued examination under 37 CFR 1.114, including the fee set forth in 37 CFR 1.17(e), was filed in this application after final rejection. Since this application is eligible for continued examination under 37 CFR 1.114, and the fee set forth in 37 CFR 1.17(e) has been timely paid, the finality of the previous Office action has been withdrawn pursuant to 37 CFR 1.114. Applicant's submission filed on May 18, 2009 has been entered.

2. Applicant's amendments dated May 18, 2009, responding to the Final Office action mailed March 18, 2009 provided in the rejection of claims 1-3, 7, 9-26, and 28-35, wherein claims 1,-3, 9-11, 13-15, 18, 20-23, and 28-35 have been amended; claims 7, 17, 19, 24-26 have been canceled; and claim 36 has been newly added.

Claims 1-3, 9-16, 18, 20-23, and 28-36 remain pending in the application and which have been fully considered by the examiner.

Applicant's arguments with respect to claims currently amended have been fully considered but are moot in view of the new grounds of rejection – see *DeKoning* et al., *Moore* et al., and *Zimmer* et al. - arts made of record, as applied here.

Claim Rejections – 35 USC § 103(a)

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

3. Claims 1-3, 9, 10, 12-16, 20-22, 26, and 28-30 are rejected under 35 U.S.C. 103(a) as being unpatentable over Talati (Pub. No. US 2004/0044997 A1) (hereinafter 'Talati') in view of DeKoning et al. (Pat. No. 6,085,333) (hereinafter 'DeKoning' art made of record), Moore et al. (Pub. No. US 2003/0092438 A1) (hereinafter 'Moore' - art made of record) and Hiller (Pat. No. US 6,658,659 B2) (hereinafter 'Hiller')

4. **As to claim 1** (Currently Amended), Talati discloses an apparatus for updating a code image (e.g., Fig. 2), comprising:

a processor executing executable code stored on a main memory occupied by and used by an old code image and a temporary memory separate from the main memory, the executable code comprising:

- a loader stored in the main memory and loading a new code image (e.g., [0014], lines 2-3; Fig. 2, element 102; [0047], lines 1-2) into the temporary memory (e.g., Fig. 1, elements 110 – Runtime Area (i.e., the main memory); 108 – Shadow Area (i.e., the temporary memory); [0014] - ... a non-disruptive code load apparatus includes means for <u>staging a new version of executable code</u> ...);
- a branch module stored in the main memory causing the processor to execute a bootstrap module within the new code image (e.g., [0032] - ... Once the new

version 102 is loaded, the system 104 and its firmware is requested to switch to the new code ...); and

- a copy module copying the new code image into the main memory space occupied by the old code image (e.g., Fig. 2, element 202 - copier; Fig. 3, element 302; [0013], lines 1-3; [0014] - ... means for transferring the executable code to a runtime area ...)

Further, Talati discloses a method and apparatus for downloading a new version of an executable code onto a system without the need to bring the system to a halt, thereby sacrificing system up time (e.g., [0005]) but does not explicitly disclose other limitations stated below.

However, in an analogous art of *Method and Apparatus for Synchronization of Code in Redundant Controllers in a Swappable Environment*, Dekoning discloses:

- the bootstrap module identifying incompatibilities between the old code image and the new code image from a difference in initialization requirements (e.g., Col. 1, Lines 50-67 - ... Incompatibility is a result of the foreign controller operating a version of software with configuration parameters not compatible to the version of the native controller because of software and configuration parameter updates; Col. 11, Lines 24-35 - ... The cache synch request will modify the cache configuration to match that of the native controller and the cache purge request initializes the spare controller's cache memory);
- the bootstrap module identifying incompatibilities between the old code image and the new code image from version information (e.g., Fig. 2, steps 230 –

Compare Firmware Revision Between Spare and Native Controllers; 240 – Firmware Compatible?; Col. 8, Lines 48-53 - ... if the revision of operating codes are incompatible, the spare controller requests synchronization 240); and

- reconciling incompatibilities by changing an initialization order (e.g., Col. 10, Lines 24-32 - ... a request to the spare controller reconfigures the spare controller's cache memory 116.1 so that cache memory 116.1 is similar to the native controller's configuration ... sends a cache purge request to the spare controller 118.2 to initialize its cache memory 330 ...;)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Dekoning into the Talati's system to further provide other limitations stated above in the Talati system.

The motivation is that it would further enhance the Talati's system by taking, advancing and/or incorporating the Dekoning's system which offers significant advantages that the spare controller firmware is compatible with the native controller's firmware using logical processes; thereby eliminating the need for a user to manually modify the spare controller's operation code and eliminating the need for any hardwired reset lines to reset the spare controller as once suggested by Dekoning (e.g., Col. 12, Lines 22-31)

Furthermore, Dekoning discloses synchronizing operating code in redundant disk storage subsystem controllers in a disk storage subsystem (e.g., Col. 1, Lines 7-10) but Talati and Dekoning do not explicitly disclose other limitations stated below.

However, in an analogous art of *Method and Apparatus for Stabilizing calls During a System Upgrade or Downgrade*, Moore discloses:

- converting a format of a data structure of the old code image to a format compatible with a data structure of the new code image, and associating persistent data of the old code image with the new code image, such that the persistent data is available in response to execution of a run-time segment of the new code image (e.g., [0005] - … ensures stable call processing during system updates must be <u>capable of converting state data to be compatible with the new service release</u>; [0024] – determines if the replicate state data needs to be converted (i.e., the new application is an upgrade) ... converts the data to the new version format ...)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Moore into the Talati-Dekoning's system to further provide other limitations stated above in the Talati-Dekoning system.

The motivation is that it would further enhance the Talati-Dekoning's system by taking, advancing and/or incorporating the Moore's system which offers significant advantages of a stabilization technology during an upgrade or downgrade of services by reducing the complications associated with checkpointing state data as once suggested by Moore (e.g., [0008])

Lastly, Talati, Dekoning, and Moore do not disclose the limitations stated below.

However, in an analogous art of *Compatible Version Module Loading*, Hiller discloses:

- the bootstrap module identifying incompatibilities between the old code image and the new code image from a difference in size and location between the old code image and the new code image (e.g., Col. 6, Lines 12-26 – <u>these different versions perform the same function during execution, but may do so in slightly different way</u>, use different arguments, and or product different results ... different versions may have <u>slightly different API function calls, performing additional supplemental functions not provided in an earlier versions</u> …); and
- accessing capability information for the old code image and capability information for the new code image and identifying a difference between the capability information (e.g., Fig. 2A, elements 208, 210, 212, and 206 - compatibility vector; Col. 8, Lines 31-48 - … The compatibility vector 206 contains the names and version of the programs that the module 200 interacts with via call. Preferably, the compatibility vector 206 identifies every program and allowable version levels for each program to be resolved and implicitly loaded for the module 200 … its respective allowable version levels that the module 200 interacts with …)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Hiller into the Talati-Dekoning-Moore's system to further provide other limitations stated above in the Talati-Dekoning-Moore system.

The motivation is that the system wherein <u>the version aware bootstrap code</u> will <u>check and ensure</u> that loaded software modules are <u>compatible with</u> one another and will therefore <u>execute properly</u> as once suggested by Hill (i.e. Abstract, Lines 10-13)

5. **As to claim 2** (Currently Amended) (incorporating the rejection in claim 1), Talati discloses the apparatus wherein the old code image is updated concurrently with normal execution of transactions by the apparatus (e.g., [0006]; [0008], lines 5-12]; [0011])

6. **As to claim 3** (Currently Amended) (incorporating the rejection in claim 1), Talati discloses the apparatus, the executable code further comprising an initialization module initiating execution of the run-time segment (e.g., [0014])

7. **As to claim 9** (Currently Amended) (incorporating the rejection in claim 31), Dmitriev discloses the apparatus wherein identifying incompatibilities comprises identifying a difference between the format of the data structures used by the old code image and the format compatible with the data structures used by the new code image (e.g., Sec. 2.2 Requirements for the PJama Evolution Technology, step 2 – For each changed class, <u>check if the format of instances that it defines became different. If so, perform conversion of instances of this class</u> …)

8. **As to claim 10** (Currently Amended), Talati discloses an apparatus for updating a code image (e.g., Fig. 2, copier copies new code), comprising:

a processor executing executable code stored on a main memory occupied by and used by an old code image and a temporary memory separate from the memory, the executable code comprising

- a loader loading a new code image (e.g., [0014], lines 2-3; Fig. 2, element 102; [0047], lines 1-2) into the temporary memory (e.g., Fig. 1, elements 110 – Runtime Area (i.e., the main memory); 108 – Shadow Area (i.e., the temporary memory); [0014] - … a non-disruptive code load apparatus includes means for staging a new version of executable code …);
- a branch module stored in the main memory causing the processor to execute a bootstrap module within the new code image (e.g., [0032] - … Once the new version 102 is loaded, the system 104 and its firmware is requested to switch to the new code …); and
- copying the new code image into the memory space occupied by the old code image (e.g., Fig. 2, element 202 - copier; Fig. 3, element 302; [0013], lines 1-3; [0014] - … means for transferring the executable code to a runtime area …)

Further, Talati discloses a method and apparatus for downloading a new version of an executable code onto a system without the need to bring the system to a halt, thereby sacrificing system up time (e.g., [0005]) but does not explicitly disclose other limitations stated below.

However, in an analogous art of *Method and Apparatus for Synchronization of Code in Redundant Controllers in a Swappable Environment*, Dekoning discloses:

- the bootstrap module identifying incompatibilities between the old code image and the new code image from a difference in initialization requirements (e.g., Col. 1, Lines 50-67 - … Incompatibility is a result of the foreign controller operating a version of software with configuration parameters not compatible to the version of the native controller because of software and configuration parameter updates; Col. 11, Lines 24-35 - ... The cache synch request will modify the cache configuration to match that of the native controller and the cache purge request initializes the spare controller's cache memory);

- a logic module configured to identify incompatibilities between the old code image and the new code image from version information (e.g., Fig. 2, steps 230 – Compare Firmware Revision Between Spare and Native Controllers; 240 – Firmware Compatible?; Col. 8, Lines 48-53 - ... if the revision of operating codes are incompatible, the spare controller requests synchronization 240); and
- reconciling incompatibilities by changing an initialization order (e.g., Col. 10, Lines 24-32 - ... a request to the spare controller reconfigures the spare controller's cache memory 116.1 so that cache memory 116.1 is similar to the native controller's configuration ... sends a cache purge request to the spare controller 118.2 to initialize its cache memory 330 ...)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Dekoning into the Talati's system to further provide other limitations stated above in the Talati system.

The motivation is that it would further enhance the Talati's system by taking, advancing and/or incorporating the Dekoning's system which offers significant advantages that the spare controller firmware is compatible with the native controller's firmware using logical processes; thereby eliminating the need for a user to manually modify the spare controller's operation code and eliminating the need for any hardwired reset lines to reset the spare controller as once suggested by Dekoning (e.g., Col. 12, Lines 22-31)

Further, Dekoning discloses synchronizing operating code in redundant disk storage subsystem controllers in a disk storage subsystem (e.g., Col. 1, Lines 7-10) but Talati and Dekoning do not explicitly disclose other limitations stated below.

However, in an analogous art of *Method and Apparatus for Stabilizing calls During a System Upgrade or Downgrade*, Moore discloses:

- converting a data structure of the old code image to a format compatible with a data structure of the new code image, and associating persistent data of the old code image with the new code image, such that the persistent data is available in response to execution of a run-time segment of the new code image (e.g., [0005] - ... ensures stable call processing during system updates must be <u>capable of converting state data to be compatible with the new service release</u>; [0024] – determines if the replicate state data needs to be converted (i.e., the new application is an upgrade) ... converts the data to the new version format ...);

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Moore into the Talati-Dekoning's system to further provide other limitations stated above in the Talati-Dekoning system.

The motivation is that it would further enhance the Talati-Dekoning's system by taking, advancing and/or incorporating the Moore's system which offers significant advantages of a stabilization technology during an upgrade or downgrade of services by reducing the complications associated with checkpointing state data as once suggested by Moore (e.g., [0008])

Lastly, Talati, Dekoning, and Moore do not disclose the limitations stated below.

However, in an analogous art of *Compatible Version Module Loading*, Hiller discloses:

- a difference in size and location between the old code image and the new code image (e.g., Col. 6, Lines 12-26 – these different versions perform the same function during execution, but may do so in slightly different way, use different arguments, and or product different results ... different versions may have slightly different API function calls, performing additional supplemental functions not provided in an earlier versions …); and
- by accessing capability information for the old code image and capability information for the new code image and identifying a difference between the capability information (e.g., Fig. 2A, elements 208, 210, 212, and 206 - compatibility vector; Col. 8, Lines 31-48 - … The compatibility vector 206 contains the names and version of the programs that the module 200 interacts with via call. Preferably, the compatibility vector 206 identifies every program and allowable version levels for each program to be resolved and implicitly loaded for the module 200 … its respective allowable version levels that the module 200 interacts with …)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Hiller into the Talati-Dekoning-Moore's system to further provide other limitations stated above in the Talati-Dekoning-Moore system.

The motivation is that the system wherein the version aware bootstrap code will check and ensure that loaded software modules are compatible with one another and will therefore execute properly as once suggested by Hill (i.e. Abstract, Lines 10-13)

9. **As to claim 12** (Previously Presented) (incorporating the rejection in claim 10), please refer to claim **9** above, accordingly.

10. **As to claim 13** (Currently Amended), Talati discloses a system that overlays an old code image with a new code image with minimal interruption of operations being performed by execution of the old code image, the system comprising:

a main memory storing an old code image (e.g., [0014], lines 2-3; Fig. 2, element 102; [0047], lines 1-2) and a temporary memory separate from the main memory and storing a new code image (e.g., Fig. 1, elements 110 – Runtime Area (i.e., the main memory); 108 – Shadow Area (i.e., the temporary memory); [0014] - … a non-disruptive code load apparatus includes means for staging a new version of executable code …);

a processor executing executable code of the old code image and the new code image, the executable code comprising:

- a loader stored in the main memory and loading the new code image (e.g., [0014], lines 2-3; Fig. 2, element 102; [0047], lines 1-2) into the temporary memory (e.g., Fig. 1, elements 110 – Runtime Area (i.e., the main memory); 108 – Shadow Area (i.e., the temporary memory); [0014] - … a non-disruptive code load apparatus includes means for <u>staging a new version of executable code</u> …); and
- a branch module stored in the main memory causing the processor to execute a bootstrap module within the new code image (e.g., [0032] - … Once the new version 102 is loaded, t<u>he system 104 and its firmware is requested to switch to the new code</u> …);

Further, Talati discloses a method and apparatus for downloading a new version of an executable code onto a system without the need to bring the system to a halt, thereby sacrificing system up time (e.g., [0005]) but does not explicitly disclose other limitations stated below.

However, in an analogous art of *Method and Apparatus for Synchronization of Code in Redundant Controllers in a Swappable Environment*, Dekoning discloses:

- the bootstrap module identifying incompatibilities between the old code image and the new code image from version information, a difference in initialization requirements (e.g., Col. 1, Lines 50-67 - … <u>Incompatibility</u> is a result of the foreign controller operating a version of software with configuration parameters not compatible to the version of the native controller because of <u>software and configuration parameter updates</u>; Col. 11, Lines 24-35 - … The cache synch request will modify the cache configuration to match that of the native controller and the cache purge request <u>initializes the spare controller's cache memory</u>);

- identifying incompatibilities between the old code image and the new code image from version information (e.g., Fig. 2, steps 230 – Compare Firmware Revision Between Spare and Native Controllers; 240 – Firmware Compatible?; Col. 8, Lines 48-53 - … if the revision of operating codes are <u>incompatible</u>, the spare controller requests synchronization 240); and
- reconciling the incompatibilities by changing an initialization order (e.g., Col. 10, Lines 24-32 - … a request to the spare controller <u>reconfigures the spare controller's cache memory</u> 116.1 so that cache memory 116.1 is similar to the native controller's configuration … sends a cache purge request to the spare controller 118.2 <u>to initialize its cache memory</u> 330 …;)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Dekoning into the Talati's system to further provide other limitations stated above in the Talati system.

The motivation is that it would further enhance the Talati's system by taking, advancing and/or incorporating the Dekoning's system which offers significant advantages that the spare controller firmware <u>is compatible with</u> the native controller's firmware using logical processes; thereby eliminating the need for a user to manually modify the spare controller's operation code and eliminating the need for any hardwired reset lines to reset the spare controller as once suggested by Dekoning (e.g., Col. 12, Lines 22-31)

Furthermore, Dekoning discloses synchronizing operating code in redundant disk storage subsystem controllers in a disk storage subsystem (e.g., Col. 1, Lines 7-10) but Talati and Dekoning do not explicitly disclose other limitations stated below.

However, in an analogous art of *Method and Apparatus for Stabilizing calls During a System Upgrade or Downgrade*, Moore discloses:

- converting a format of a data structure of the old code image to a format compatible with a data structure of the new code image, and associating persistent data of the old code image with the new code image, such that the persistent data is available in response to execution of a run-time segment of the new code image (e.g., [0005] - ... ensures stable call processing during system updates must be <u>capable of converting state data to be compatible with the new service release</u>; [0024] – determines if the replicate state data needs to be converted (i.e., the new application is an upgrade) ... converts the data to the new version format ...)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Moore into the Talati-Dekoning's system to further provide other limitations stated above in the Talati-Dekoning system.

The motivation is that it would further enhance the Talati-Dekoning's system by taking, advancing and/or incorporating the Moore's system which offers significant advantages of a stabilization technology during an upgrade or downgrade of services by reducing the complications associated with checkpointing state data as once suggested by Moore (e.g., [0008]) Lastly, Talati, Dekoning, and Moore do not disclose the limitations stated below. However, in an analogous art of *Compatible Version Module Loading*, Hiller discloses:

- the bootstrap module identifying incompatibilities between the old code image and the new code image from version information, a difference in size and location between the old code image and the new code image (e.g., Col. 6, Lines 12-26 – these different versions perform the same function during execution, but may do so in slightly different way, use different arguments, and or product different results ... different versions may have slightly different API function calls, performing additional supplemental functions not provided in an earlier versions …); and
- accessing capability information for the old code image and capability information for the new code image and identifying a difference between the capability information (e.g., Fig. 2A, elements 208, 210, 212, and 206 - compatibility vector; Col. 8, Lines 31-48 - … The compatibility vector 206 contains the names and version of the programs that the module 200 interacts with via call. Preferably, the compatibility vector 206 identifies every program and allowable version levels for each program to be resolved and implicitly loaded for the module 200 … its respective allowable version levels that the module 200 interacts with …)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Hiller into the Talati-Dekoning-Moore's system to further provide other limitations stated above in the Talati-Dekoning-Moore system.

The motivation is that the system wherein the version aware bootstrap code will check and ensure that loaded software modules are compatible with one another and will therefore execute properly as once suggested by Hill (i.e. Abstract, Lines 10-13)

11. **As to claim 14** (Currently Amended) (incorporating the rejection in claim 13), Hiller discloses the system, the executable code further comprising a copy module stored within the new code image overlaying the new code image in main memory with the old code image in response to reconciling the incompatibilities (e.g., Fig. 2A, element 206; Fig. 2B; Col. 3, Lines 42-46; Col. 4, Lines 64-67))

12. **As to claim 15** (Currently Amended) (incorporating the rejection in claim 14), Talati discloses the system, the loader loading the new code image into the temporary memory in response to an interrupt (e.g., Fig. 3, step 302; [0040]; [0041])

13. **As to claim 16** (original) (incorporating the rejection in claim 15), Talati discloses the system wherein the update module stores the old code image pointer (e.g., Fig. 2, elements 204, 208, and 212; [0023], lines 4-10) and the new code image pointer (e.g., Fig. 2, elements 206, 210, and 214; [0023], lines 11-16) in the data structure.

14. **As to claim 20** (Currently Amended), Talati discloses a method for updating a code image (e.g., Fig. 2, Copier copies new code), comprising:

- loading, by use of a processor, a new code image into a temporary memory location (e.g., [0014], lines 2-3; Fig. 2, element 102; [0047], lines 1-2) separate from a main memory space (e.g., Fig. 1, element 110; [0009], line 2, runtime area) occupied by and used by an old code image (e.g., Fig. 1, elements 110 – Runtime Area (i.e., the main memory); 108 – Shadow Area (i.e., the temporary memory); [0014] - … a non-disruptive code load apparatus includes means for <u>staging a new version of executable code</u> …);
- executing a bootstrap module within the new code image (e.g., [0032] - … Once the new version 102 is loaded, t<u>he system 104 and its firmware is requested to switch to the new code</u> …); and
- copying the new code image into the main memory space occupied by the old code image (e.g., Fig. 2, element 202 - copier; Fig. 3, element 302; [0013], lines 1-3; [0014] - … means for <u>transferring the executable code to a runtime area</u> …)

Further, Talati discloses a method and apparatus for downloading a new version of an executable code onto a system without the need to bring the system to a halt, thereby sacrificing system up time (e.g., [0005]) but does not explicitly disclose other limitations stated below.

However, in an analogous art of *Method and Apparatus for Synchronization of Code in Redundant Controllers in a Swappable Environment*, Dekoning discloses:

- identifying, using the bootstrap module executed by the processor, incompatibilities between the old code image and the new code image from a difference in initialization requirements (e.g., Col. 1, Lines 50-67 - …

Incompatibility is a result of the foreign controller operating a version of software with configuration parameters not compatible to the version of the native controller because of software and configuration parameter updates; Col. 11, Lines 24-35 - … The cache synch request will modify the cache configuration to match that of the native controller and the cache purge request initializes the spare controller's cache memory);

- identifying, using the bootstrap module executed by the processor, incompatibilities between the old code image and the new code image from version information (e.g., Fig. 2, steps 230 – Compare Firmware Revision Between Spare and Native Controllers; 240 – Firmware Compatible?; Col. 8, Lines 48-53 - … if the revision of operating codes are incompatible, the spare controller requests synchronization 240); and
- reconciling, using the bootstrap module executed by the processor, the incompatibilities by changing an initialization order (e.g., Col. 10, Lines 24-32 - … a request to the spare controller reconfigures the spare controller's cache memory 116.1 so that cache memory 116.1 is similar to the native controller's configuration … sends a cache purge request to the spare controller 118.2 to initialize its cache memory 330 …;)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Dekoning into the Talati's system to further provide other limitations stated above in the Talati system.

The motivation is that it would further enhance the Talati's system by taking, advancing and/or incorporating the Dekoning's system which offers significant advantages that the spare controller firmware <u>is compatible with</u> the native controller's firmware using logical processes; thereby eliminating the need for a user to manually modify the spare controller's operation code and eliminating the need for any hardwired reset lines to reset the spare controller as once suggested by Dekoning (e.g., Col. 12, Lines 22-31)

Furthermore, Dekoning discloses synchronizing operating code in redundant disk storage subsystem controllers in a disk storage subsystem (e.g., Col. 1, Lines 7-10) but Talati and Dekoning do not explicitly disclose other limitations stated below.

However, in an analogous art of *Method and Apparatus for Stabilizing calls During a System Upgrade or Downgrade*, Moore discloses:

- and converting a data structure of the old code image to a format compatible with a data structure of the new code image using bootstrap code of the new code image, and associating persistent data of the old code image with the new code image, such that the persistent data is available in response to execution of a run-time segment of the new code image (e.g., [0005] - ... ensures stable call processing during system updates must be <u>capable of converting state data to be compatible with the new service release</u>; [0024] – determines if the replicate state data needs to be converted (i.e., the new application is an upgrade) ... converts the data to the new version format ...)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Moore into the Talati-Dekoning's system to further provide other limitations stated above in the Talati-Dekoning system.

The motivation is that it would further enhance the Talati-Dekoning's system by taking, advancing and/or incorporating the Moore's system which offers significant advantages of a stabilization technology during an upgrade or downgrade of services by reducing the complications associated with checkpointing state data as once suggested by Moore (e.g., [0008])

Lastly, Talati, Dekoning, and Moore do not disclose the limitations stated below.

However, in an analogous art of *Compatible Version Module Loading*, Hiller discloses:

- a difference in size and location between the old code image and the new code image (e.g., Col. 6, Lines 12-26 – these different versions perform the same function during execution, but may do so in slightly different way, use different arguments, and or product different results ... different versions may have slightly different API function calls, performing additional supplemental functions not provided in an earlier versions ...); and
- accessing capability information for the old code image and capability information for the new code image and identifying a difference between the capability information (e.g., Fig. 2A, elements 208, 210, 212, and 206 - compatibility vector; Col. 8, Lines 31-48 - ... The compatibility vector 206 contains the names and version of the programs that the module 200

> interacts with via call. Preferably, the compatibility vector 206 identifies every program and allowable version levels for each program to be resolved and implicitly loaded for the module 200 … its respective allowable version levels that the module 200 interacts with …)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Hiller into the Talati-Dekoning-Moore's system to further provide other limitations stated above in the Talati-Dekoning-Moore system.

The motivation is that the system wherein <u>the version aware bootstrap code</u> will <u>check and ensure</u> that loaded software modules are <u>compatible with</u> one another and will therefore <u>execute properly</u> as once suggested by Hill (i.e. Abstract, Lines 10-13)

15. **As to claim 21** (Currently Amended) (incorporating the rejection in claim 20), Talati discloses the method wherein the new code image is copied to the main memory concurrently with execution of regular computer operations (e.g., [0011]; [0014])

16. **As to claim 22** (Currently Amended) (incorporating the rejection in claim 20), Talati discloses the method further comprising initiating execution of a run-time segment of the new code image (e.g., [0049])

17. **As to claim 28** (Currently Amended) (incorporating the rejection in claim 20), please refer to claim **9** above, accordingly.

18. **As to claim 29** (Currently Amended), Talati discloses an apparatus for updating a code image (e.g., [0014], lines 2-3; Fig. 2, copier copies new code; [0047], lines 1-2), the apparatus comprising:

a processor executing executable code stored on a main memory occupied by and used by an old code image and a temporary memory separate form the main memory, the executable code comprising:

- means for loading a new code image (e.g., [0014], lines 2-3; Fig. 2, element 102; [0047], lines 1-2) into the temporary memory (e.g., Fig. 1, elements 110 – Runtime Area (i.e., the main memory); 108 – Shadow Area (i.e., the temporary memory); [0014] - … a non-disruptive code load apparatus includes means for <u>staging a new version of executable code</u> …);
- means for causing the processor to execute a bootstrap module within the new code image (e.g., [0032] - … Once the new version 102 is loaded, t<u>he system 104 and its firmware is requested to switch to the new code</u> …); and
- means for copying the new code image into the memory space occupied by the old code image (e.g., Fig. 2, element 202 - copier; Fig. 3, element 302; [0013], lines 1-3; [0014] - … means for <u>transferring the executable code to a runtime area</u> …)

Further, Talati discloses a method and apparatus for downloading a new version of an executable code onto a system without the need to bring the system to a halt, thereby sacrificing system up time (e.g., [0005]) but does not explicitly disclose other limitations stated below.

However, in an analogous art of *Method and Apparatus for Synchronization of Code in Redundant Controllers in a Swappable Environment*, Dekoning discloses:

- means within the bootstrap module for identifying incompatibilities between the old code image and the new code image from a difference in initialization requirements (e.g., Col. 1, Lines 50-67 - ... <u>Incompatibility</u> is a result of the foreign controller operating a version of software with configuration parameters not compatible to the version of the native controller because of <u>software and configuration parameter updates</u>; Col. 11, Lines 24-35 - ... The cache synch request will modify the cache configuration to match that of the native controller and the cache purge request <u>initializes the spare controller's cache memory</u>);
- means within the bootstrap module for identifying incompatibilities between the old code image and the new code image from version information (e.g., Fig. 2, steps 230 – Compare Firmware Revision Between Spare and Native Controllers; 240 – Firmware Compatible?; Col. 8, Lines 48-53 - ... if the revision of operating codes are <u>incompatible</u>, the spare controller requests synchronization 240); and
- means within the bootstrap module for reconciling the incompatibilities by changing an initialization order (e.g., Col. 10, Lines 24-32 - ... a request to the spare controller <u>reconfigures the spare controller's cache memory</u>

116.1 so that cache memory 116.1 is similar to the native controller's configuration … sends a cache purge request to the spare controller 118.2 <u>to initialize its cache memory</u> 330 …;)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Dekoning into the Talati's system to further provide other limitations stated above in the Talati system.

The motivation is that it would further enhance the Talati's system by taking, advancing and/or incorporating the Dekoning's system which offers significant advantages that the spare controller firmware <u>is compatible with</u> the native controller's firmware using logical processes; thereby eliminating the need for a user to manually modify the spare controller's operation code and eliminating the need for any hardwired reset lines to reset the spare controller as once suggested by Dekoning (e.g., Col. 12, Lines 22-31)

Furthermore, Dekoning discloses synchronizing operating code in redundant disk storage subsystem controllers in a disk storage subsystem (e.g., Col. 1, Lines 7-10) but Talati and Dekoning do not explicitly disclose other limitations stated below.

However, in an analogous art of *Method and Apparatus for Stabilizing calls During a System Upgrade or Downgrade*, Moore discloses:

- and converting a format of a data structure of the old code image to a format compatible with a data structure of the new code image using bootstrap code of the new code image, and associating persistent data of the old code image with the new code image, such that the persistent data is available in response to execution of a run0time segment of the new code image (e.g., [0005] - … ensures stable call processing during system updates must be <u>capable of converting state data to be compatible with the new service release</u>; [0024] – determines if the replicate state data needs to be converted (i.e., the new application is an upgrade) ... converts the data to the new version format ...)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Moore into the Talati-Dekoning's system to further provide other limitations stated above in the Talati-Dekoning system.

The motivation is that it would further enhance the Talati-Dekoning's system by taking, advancing and/or incorporating the Moore's system which offers significant advantages of a stabilization technology during an upgrade or downgrade of services by reducing the complications associated with checkpointing state data as once suggested by Moore (e.g., [0008])

Lastly, Talati, Dekoning, and Moore do not explicitly disclose the limitations stated below.

However, in an analogous art of *Compatible Version Module Loading*, Hiller discloses:

- means within the bootstrap module for identifying incompatibilities between the old code image and the new code image from a difference in size and location between the old code image and the new code image (e.g., Col. 6, Lines 12-26 – <u>these different versions perform the same</u>

function during execution, but may do so in slightly different way, use different arguments, and or product different results ... different versions may have slightly different API function calls, performing additional supplemental functions not provided in an earlier versions ...); and

- accessing capability information for the old code image and capability information for the new code image and identifying a difference between the capability information (e.g., Fig. 2A, elements 208, 210, 212, and 206 - compatibility vector; Col. 8, Lines 31-48 - ... The compatibility vector 206 contains the names and version of the programs that the module 200 interacts with via call. Preferably, the compatibility vector 206 identifies every program and allowable version levels for each program to be resolved and implicitly loaded for the module 200 ... its respective allowable version levels that the module 200 interacts with ...)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Hiller into the Talati-Dekoning-Moore's system to further provide other limitations stated above in the Talati-Dekoning-Moore system.

The motivation is that the system wherein the version aware bootstrap code will check and ensure that loaded software modules are compatible with one another and will therefore execute properly as once suggested by Hill (i.e. Abstract, Lines 10-13)

19. **As to claim 30** (Currently Amended), Talati discloses an article of manufacture comprising a program storage medium readable by a processor and embodying one or more instructions executable by a processor to perform a method for updating a code image, the method comprising:

- loading a new code image into a temporary memory location separate from a memory space occupied by and used by an old code image (e.g., Fig. 1, elements 110 – Runtime Area (i.e., the main memory); 108 – Shadow Area (i.e., the temporary memory); [0014] - … a non-disruptive code load apparatus includes means for <u>staging a new version of executable code</u> …);
- causing the processor to execute a bootstrap module within the new code image (e.g., [0032] - … Once the new version 102 is loaded, t<u>he system 104 and its firmware is requested to switch to the new code</u> …); and
- copying the new code image into the main memory space occupied by the old code image (e.g., Fig. 2, element 202 - copier; Fig. 3, element 302; [0013], lines 1-3; [0014] - … means for <u>transferring the executable code to a runtime area</u> …)

Further, Talati discloses a method and apparatus for downloading a new version of an executable code onto a system without the need to bring the system to a halt, thereby sacrificing system up time (e.g., [0005]) but does not explicitly disclose other limitations stated below.

However, in an analogous art of *Method and Apparatus for Synchronization of Code in Redundant Controllers in a Swappable Environment*, Dekoning discloses:

- identifying, using the bootstrap module executed by the processor, incompatibilities between the old code image and the new code image from a difference in initialization requirements (e.g., Col. 1, Lines 50-67 - … Incompatibility is a result of the foreign controller operating a version of software with configuration parameters not compatible to the version of the native controller because of software and configuration parameter updates; Col. 11, Lines 24-35 - … The cache synch request will modify the cache configuration to match that of the native controller and the cache purge request initializes the spare controller's cache memory);
- identifying, using the bootstrap module executed by the processor, incompatibilities between the old code image and the new code image from version information (e.g., Fig. 2, steps 230 – Compare Firmware Revision Between Spare and Native Controllers; 240 – Firmware Compatible?; Col. 8, Lines 48-53 - … if the revision of operating codes are incompatible, the spare controller requests synchronization 240); and
- reconciling, using the bootstrap module executed by the processor, the incompatibilities by changing an initialization order (e.g., Col. 10, Lines 24-32 - … a request to the spare controller reconfigures the spare controller's cache memory 116.1 so that cache memory 116.1 is similar to the native controller's configuration … sends a cache purge request to the spare controller 118.2 to initialize its cache memory 330 …);

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Dekoning into the Talati's system to further provide other limitations stated above in the Talati system.

The motivation is that it would further enhance the Talati's system by taking, advancing and/or incorporating the Dekoning's system which offers significant advantages that the spare controller firmware <u>is compatible with</u> the native controller's firmware using logical processes; thereby eliminating the need for a user to manually modify the spare controller's operation code and eliminating the need for any hardwired reset lines to reset the spare controller as once suggested by Dekoning (e.g., Col. 12, Lines 22-31)

Furthermore, Dekoning discloses synchronizing operating code in redundant disk storage subsystem controllers in a disk storage subsystem (e.g., Col. 1, Lines 7-10) but Talati and Dekoning do not explicitly disclose other limitations stated below.
However, in an analogous art of *Method and Apparatus for Stabilizing calls During a System Upgrade or Downgrade*, Moore discloses:

- converting a format of a data structure of the old code image to a format compatible with a data structure of the new code image using bootstrap code of the new code image, and associating persistent data of the old code image with the new code image, such that the persistent data is available in response to execution of a run-time segment of the new code image (e.g., [0005] - … ensures stable call processing during system updates must be <u>capable of converting state data to be compatible with the new service release</u>; [0024] – determines if the replicate state data needs to be converted (i.e., the new application is an upgrade) ... converts the data to the new version format ...)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Moore into the Talati-Dekoning's system to further provide other limitations stated above in the Talati-Dekoning system.

The motivation is that it would further enhance the Talati-Dekoning's system by taking, advancing and/or incorporating the Moore's system which offers significant advantages of a stabilization technology during an upgrade or downgrade of services by reducing the complications associated with checkpointing state data as once suggested by Moore (e.g., [0008])

Lastly, Talati, Dekoning, and Moore do not disclose the limitations stated below.

However, in an analogous art of *Compatible Version Module Loading*, Hiller discloses:

- a difference in size and location between the old code image and the new code image (e.g., Col. 6, Lines 12-26 – these different versions perform the same function during execution, but may do so in slightly different way, use different arguments, and or product different results ... different versions may have slightly different API function calls, performing additional supplemental functions not provided in an earlier versions ...); and
- accessing capability information for the old code image and capability information for the new code image and identifying a difference between the capability information (e.g., Fig. 2A, elements 208, 210, 212, and 206 - compatibility vector;

Col. 8, Lines 31-48 - … The compatibility vector 206 contains the names and version of the programs that the module 200 interacts with via call. Preferably, the compatibility vector 206 identifies every program and allowable version levels for each program to be resolved and implicitly loaded for the module 200 … its respective allowable version levels that the module 200 interacts with …)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Hiller into the Talati-Dekoning-Moore's system to further provide other limitations stated above in the Talati-Dekoning-Moore system.

The motivation is that the system wherein <u>the version aware bootstrap code</u> will <u>check and ensure</u> that loaded software modules are <u>compatible with</u> one another and will therefore <u>execute properly</u> as once suggested by Hill (i.e. Abstract, Lines 10-13)

20. Claim 11, 18, and 23 are rejected under 35 U.S.C. 103(a) as being unpatentable over Talati in view of DeKoning, Moore, Hiller and E. Schwabe (Pat. No. US 6,986,132 B1) (hereinafter 'Schwabe')

21. **As to claim 11** (Currently Amended) (incorporating the rejection in claim 10), Talati, DeKoning, Moore, and Hiller do not disclose the limitations stated below.

However, in an analogous art of *Remote Incremental Program Binary Compatibility Verification Using API Definitions*, Schwabe discloses the apparatus, the executable code further comprising a copy module copying the new code image over the old code image in the main memory after the incompatibilities have been reconciled

(e.g., Fig. 17, element 1440 – version; Fig. 18, element 1470 – version; Fig. 19, element 1515 – verify version of API definition file used during verification is compatible with version of referenced binary file; Fig. 20A – 3. verify backward compatible version with content; Fig. 20C – verify versions using API definitions files, elements of 1600, 1605, and 1610)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Schwabe into the Talati-DeKoning-Moore-Hiller's system to further provide the limitations stated above in the Talati-DeKoning-Moore-Hiller system.

The motivation is that use of the verifier enables verification of a program's integrity and allows the use of an interpreter that does not execute the usual stack monitoring instructions during program execution, thereby greatly accelerating the program interpretation process as once suggested by Schwabe (i.e. Col. 13, Line 66 through Col. 14, Line 8)

22. **As to claim 18** (Currently Amended) (incorporating the rejection in claim 33), Schwabe discloses the system, wherein the bootstrap module further reconciles the incompatibilities by updating modules that interface with the new code image (e.g., Col. 9, Lines 6-11)

23. **As to claim 23** (Currently Amended) (incorporating the rejection in claim 20), Schwabe discloses the method wherein the new code image is copied into the main

memory until incompatibilities are reconciled (e.g., Col. 5, Lines 49-53; Col. 10, Lines 40-44; Col. 11, Line 58 through Col. 12, Lines 6; Figs. 15A-15B; Col. 20, Line 64 through Col. 21, Line 4, 14-20; Fig. 17; Col. 22, Lines 4-16; Figs. 20A-20D; Col. 24, Lines 24-32)

24. Claim 31-36 are rejected under 35 U.S.C. 103(a) as being unpatentable over Talati in view of DeKoning, Moore, Hiller and Zimmer et al. (Pub. No. US 2003/0120909 A1) (hereinafter 'Zimmer' - art made of record)

25. **As to claim 31** (Currently Amended) (incorporating the rejection in claim 1), Talati discloses the apparatus, wherein the loader configures the temporary memory so that the executable code is executed directly from the temporary memory (e.g., Fig. 1, elements 110 – Runtime Area (i.e., the main memory); 108 – Shadow Area (i.e., the temporary memory); [0014] - ... a non-disruptive code load apparatus includes means for staging a new version of executable code ...), the executable code further comprising an update module stored in the main memory maintaining an old code image pointer, a new code image pointer (e.g., Fig. 2, elements 110 – Run Time Area; 108 – Shadow Area; [0034] - ... illustrating the interaction between the shadow area 108 and runtime area ... shows the memory layouts for the shadow area 108 and the runtime area 110 of the system 104 ...; [0035] – [0036])

Further, Hiller discloses capability fields storing the capability information, an old code image version number, a new code image version number, the old code image pointer, the new code image pointer, the capability fields, the old code image version number, and the new code image version number used by the bootstrap module, wherein the bootstrap module follows the old code image pointer to locate an old code image header and a version field within the old code image header and follows the new code image pointer to locate a new code image header and a version field within the new code image header (e.g., Col. 6, Lines 12-26 – these different versions perform the same function during execution, but may do so in slightly different way, use different arguments, and or product different results ... different versions may have slightly different API function calls, performing additional supplemental functions not provided in an earlier versions …)

Furthermore, Dekoning discloses reconciling incompatibilities between the old code image and the code image further comprises adjusting configuration setting and parameters lists (e.g., Col. 1, Lines 50-67 - … Incompatibility is a result of the foreign controller operating a version of software with configuration parameters not compatible to the version of the native controller because of software and configuration parameter updates; Col. 11, Lines 24-35 - … The cache synch request will modify the cache configuration to match that of the native controller and the cache purge request initializes the spare controller's cache memory; Col. 10, Lines 24-32 - … a request to the spare controller reconfigures the spare controller’s cache memory 116.1 so that cache memory 116.1 is similar to the native controller’s configuration … sends a cache purge request to the spare controller 118.2 to initialize its cache memory 330 …;)

Talati, DeKoning, Moore, and Hiller do not disclose the limitations stated below.

However, in an analogous art of *Boot Process*, Zimmer discloses:

- the old code image header and the new code image header are organized according to the Microcode Reconstruct and Boot format, the bootstrap module reading the capability information from the old code image and the new code image and storing the capability information of the old code image and the new code image in the capability fields, the capability information comprising an indication that an EMULEX FLASH RAM is provided, the persistent data comprising login tables (e.g., [0006] – a boot routine initialized in a computer by bootstrapping a <u>volume top file</u> (VTF) located in a first addressable location accessible upon the initializing of the boot routine and the volume top file bootstrapping a set of firmware modules; Fig. 2;–[0015] - ... provides a mechanism to locate code and data at fixed location required by the processor architecture and to bootstrap other firmware module 44(a)-44(b) residing within firmware volume 88; [0016] – [0017]; [0023] – [0024])

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Zimmer into the Talati-DeKoning-Moore-Hiller's system to further provide the limitations stated above in the Talati-DeKoning-Moore-Hiller system.

The motivation is that it would further enhance the Talati-DeKoning-Moore-Hiller's system by taking, advancing and/or incorporating the Zimmer's system which offers significant advantages to overcome prior arts when attached device details change, the BIOS program needs to be changed, either during computer system setup or manually as once suggested by Zimmer (e.g., [0002])

26. **As to claim 32** (Currently Amended) (incorporating the rejection in claim 10), please refer to claim **31** above, accordingly.

27. **As to claim 33** (Currently Amended) (incorporating the rejection in claim 13), please refer to claim **31** above, accordingly.

28. **As to claim 34** (Currently Amended) (incorporating the rejection in claim 20), please refer to claim **31** above, accordingly.

29. **As to claim 35** (Currently Amended) (incorporating the rejection in claim 30), please refer to claim **31** above, accordingly.

30. **As to claim 36** (New) (incorporating the rejection in claim 29), please refer to claim **31** above, accordingly.

Conclusion

31. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Ben C. Wang whose telephone number is 571-270-1240. The examiner can normally be reached on Monday - Friday, 8:00 a.m. - 5:00 p.m., EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached on 571-272-3695. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/Ben C Wang/
Ben C. Wang
Examiner, Art Unit 2192

/Tuan Q. Dam/
Supervisory Patent Examiner, Art Unit 2192